Assignment 5: The Resistor Problem

Jnaneswara Rao Rompilli [EE20B052]

March 8, 2022

Abstract

- To solve for potential and currents in a system.
- To solve 2-D Laplace equations in an iterative manner.
- To plot graphs to understand the 2-D Laplace equation.

1 Introduction

A wire is soldered to the middle of a copper plate and its voltage is held at 1 Volt. One side of the plate is

Ohm's law

$$\vec{J} = \sigma \vec{E} \tag{1}$$

Charge Continuity equation.

$$\nabla . \vec{J} = -\frac{\partial \rho}{\partial t} \tag{2}$$

From the above equations,

$$\nabla^2 \phi = \frac{1}{\sigma} \frac{\partial \rho}{\partial t} \tag{3}$$

For DC currents, the right side is zero, and we obtain

$$\nabla^2 \phi = 0 \tag{4}$$

2 Tasks

2.1 Parameters

They are assigned default values, which will be corrected via commandline arguments

```
Nx = 25  # size laong x
Ny = 25  # size along y
radius = 8  # radius of central lead
Niter = 1500  # number of iterations to perform

argc = len(argv)

# Command line arguments
if argc == 5:
    Nx = argv[1]
    Ny = argv[2]
    radius = argv[3]
    Niter = argv[4]
```

2.2 Variable initialization

Create a zero 2-D array of size Nx x Ny and assign 1 to cordinates within radius 1 from center

```
# Intialize required variables
phi = np.zeros((Ny, Nx))
x = np.linspace(-0.5, 0.5, Nx)
y = np.linspace(-0.5, 0.5, Ny)

Y, X = np.meshgrid(y, x)
```

2.3 Allocating potential and plotting it

```
# Assign potential = 1
ii = np.where(X * X + Y * Y <= (0.35 * 0.35))
phi[ii] = 1

plt.plot(x[ii[0]], y[ii[1]], "or", label="V = 1")
```

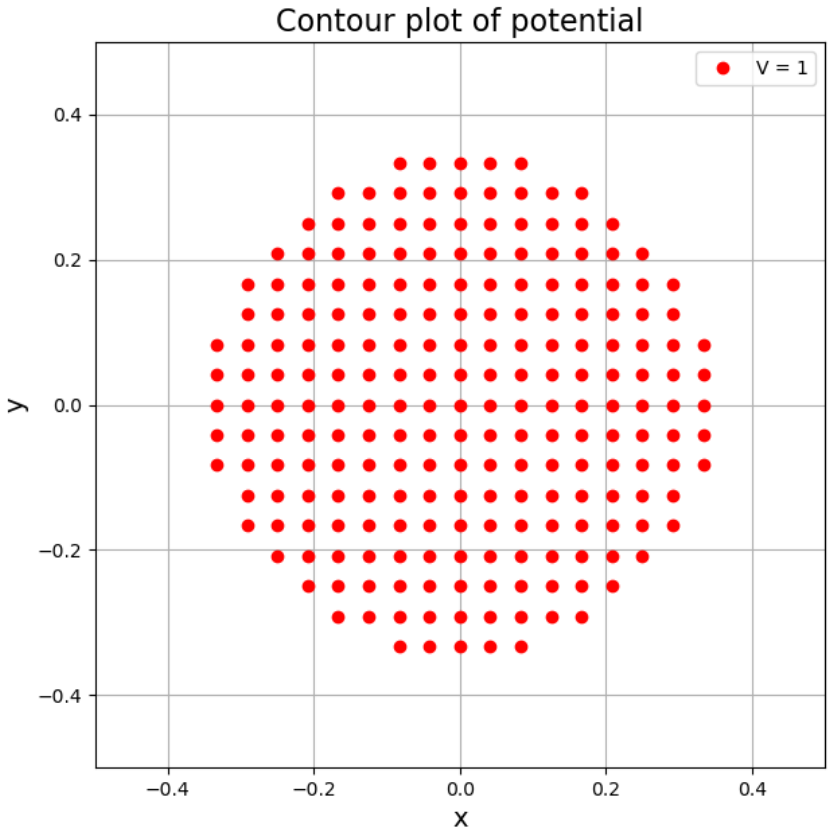

Figure 1: semilogy scale

2.4 Updating Potential

After converting the equation(4) to discrete domain, we can update matrix through iterations

$$\phi_{i,j} = \frac{\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1}}{4} \tag{5}$$

The bottom boundary is grounded. The other 3 boundaries have a normal potential of zero

```
oldphi = np.zeros((Ny, Nx))
errors = np.zeros((Niter, 1))

# Updating the potential
for k in range(Niter):
    oldphi = phi.copy()
    new = oldphi[2:, 1:-1] + oldphi[1:-1, 0:-2] + oldphi[1:-1, 2:] + oldphi[0:-2, 1:-1]
    phi[1:-1, 1:-1] = 0.25 * new

    phi[1:-1, 0] = phi[1:-1, 1]
    phi[1:-1, -1] = phi[1:-1, -2]
    phi[0, 1:-1] = phi[1, 1:-1]
    phi[ii] = 1.0

    errors[k] = (abs(phi - oldphi)).max()
```

2.5 Plotting the errors

We will plot the errors on semi-log and log-log plots. We can observe the error decreases very slow

```
plt.semilogy(k, errors, "go", markersize=2, label="errors")

plt.loglog(k, errors, "go", markersize=2, label="errors")
```

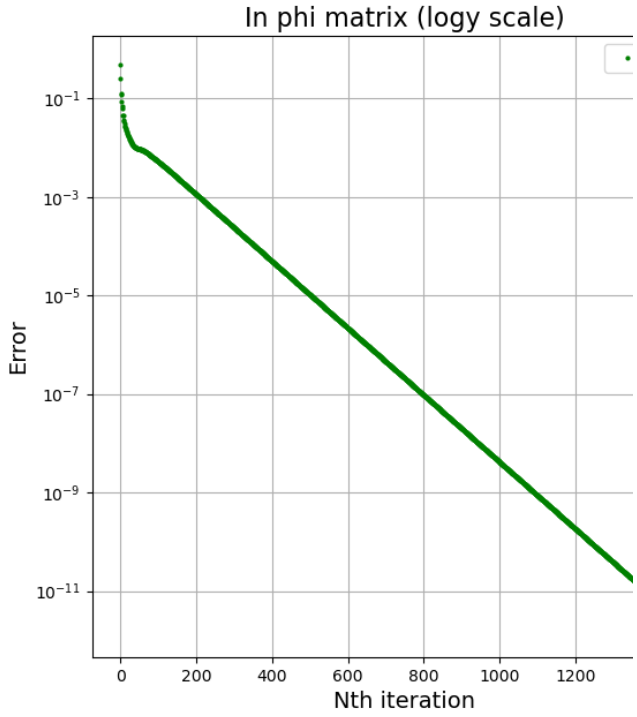

Figure 2: Error (semilog)

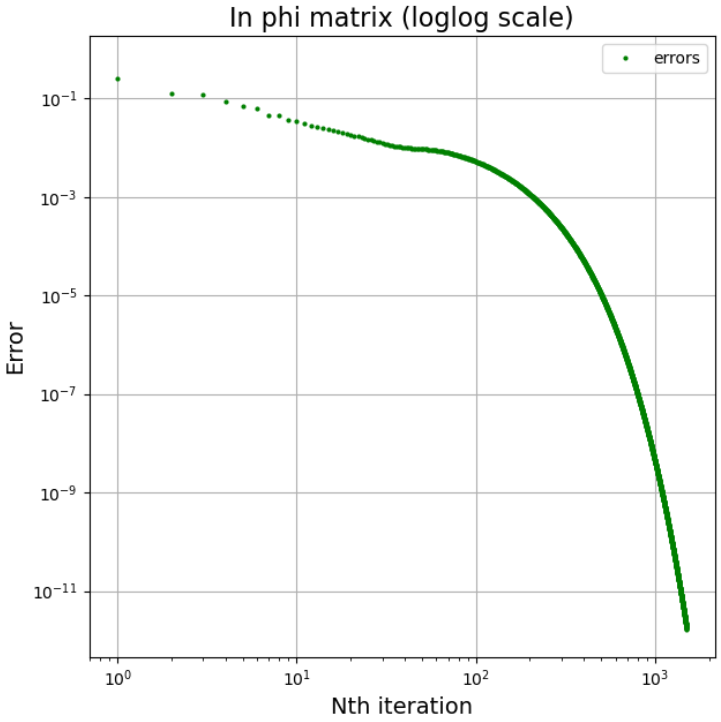

Figure 3: Error (loglog)

2.5.1 Fitting exponential curve

We observed that the error is decaying exponentially for higher iterations. We will fit two curves.

- Considering all iteration
- Considering after 500th iteration

```
# Function to fit exponential curve using lstsq
def fit_curve(Niter, start, errors):
    xfir = np.full((Niter - start, 1), fill_value=1)
    xsec = np.arange(start, Niter).reshape((Niter - start, 1))
    M = np.hstack((xfir, xsec))
    y2 = log(errors[start:])
    coeff2 = lstsq(M, y2, rcond=None)[0]
    return exp(coeff2[0]) * exp(coeff2[1] * xsec)

yfit1 = fit_curve(Niter, 0, errors)
yfit2 = fit_curve(Niter, 500, errors)
```

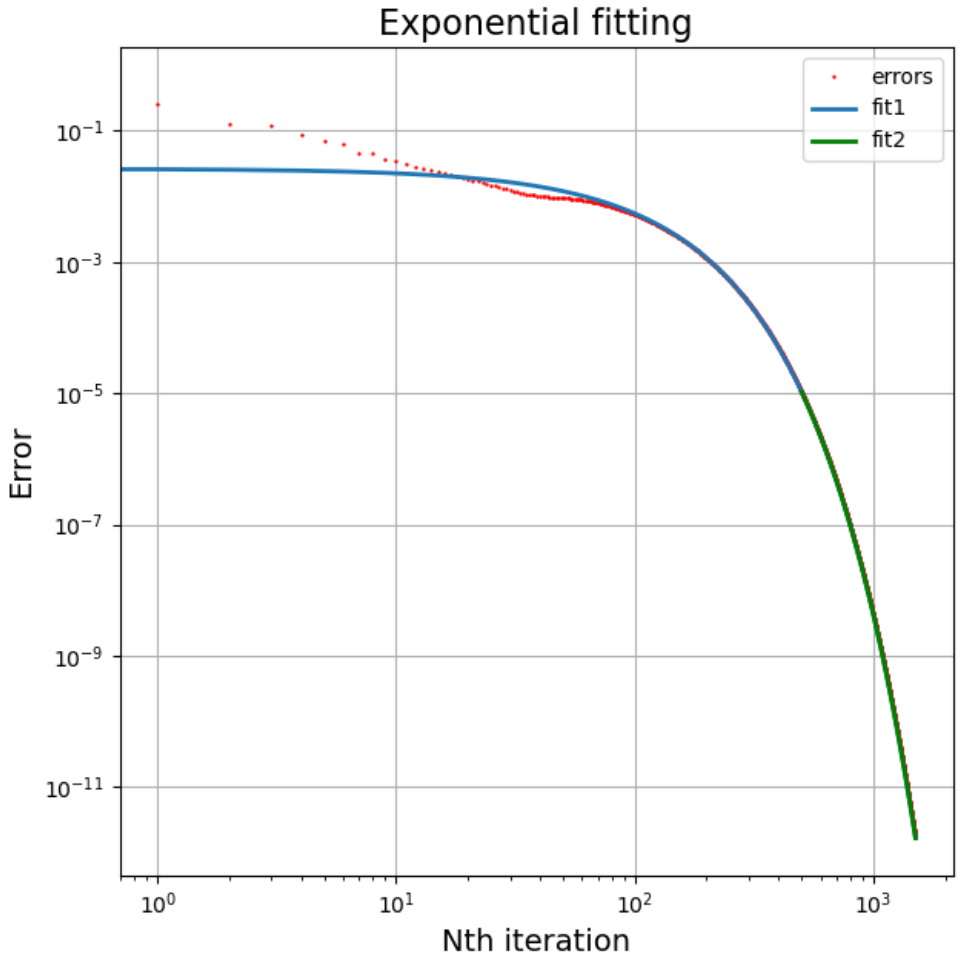

Figure 4: Potting exponential curves

2.6 Plotting Potential

```
# Contour plot of potential
plt.title("Contour plot of V", fontsize=16)
```

```
plt.contourf(Y, X[::-1], phi)
plt.plot(x[ii[0]], y[ii[1]], "or", label="V = 1")

# Surface plot of potential
fig8 = figure(8, figsize=(7, 7))
ax = p3.Axes3D(fig8, auto_add_to_figure=False)
fig8.add_axes(ax)
plt.title("The 3-D surface plot of the potential", fontsize=16)
surface = ax.plot_surface(Y, X, phi.T, rstride=1, cstride=1, cmap=cm.jet)
```

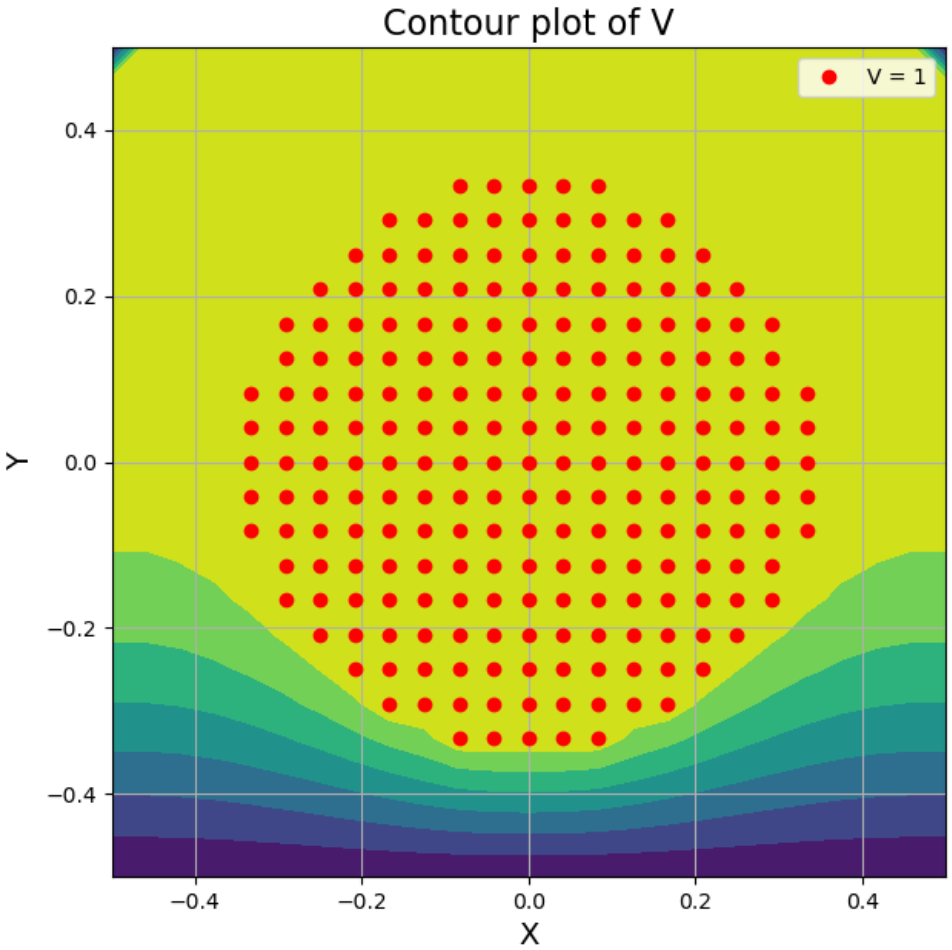

Figure 5: Contour plot of potential

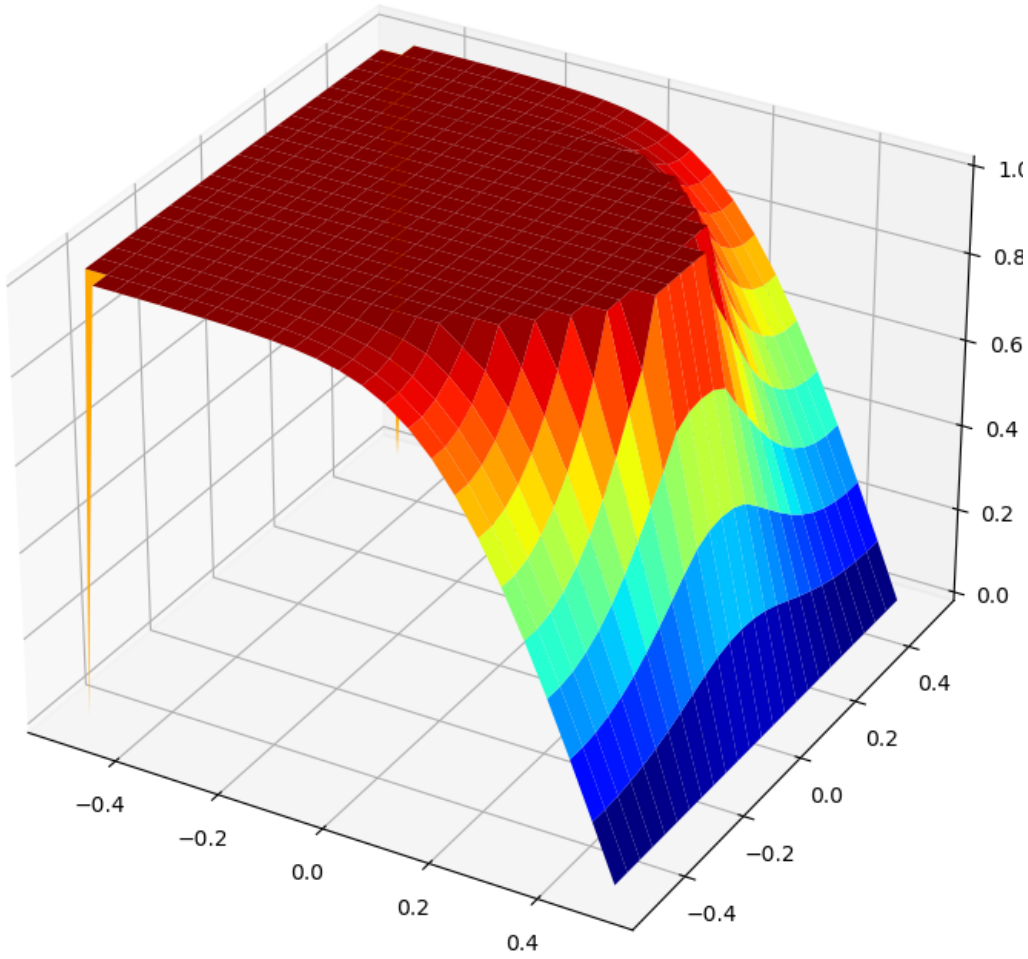

Figure 6: 3D Potential plot

2.7 Vector plot of currents

We use the below equations to calculate current,

$$J_{x,ij} = \frac{1}{2}(\phi_{i,j-1} - \phi_{i,j+1}) \tag{6}$$

$$J_{y,ij} = \frac{1}{2}(\phi_{i-1,j} - \phi_{i+1,j}) \tag{7}$$

```
Jx, Jy = (
    1 / 2 * (phi[1:-1, 0:-2] - phi[1:-1, 2:]),
    1 / 2 * (phi[:-2, 1:-1] - phi[2:, 1:-1]),
)

# Vector plot of currents
plt.quiver(Y[1:-1, 1:-1], -X[1:-1, 1:-1], -Jx[:, ::-1], -Jy)
plt.plot(x[ii[0]], y[ii[1]], "or", label="V = 1")
```

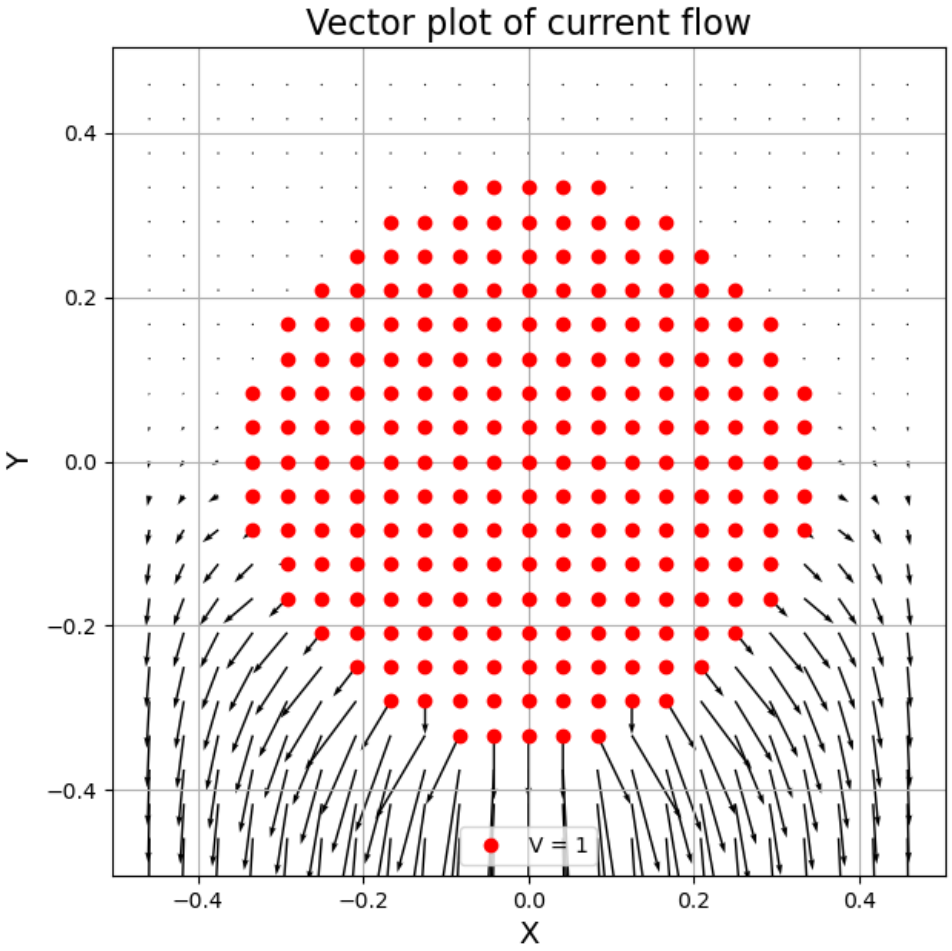

Figure 7: Current vectors

3 Conclusion

- Most of the current is restricted to bottom part of the wire and it is normal to the surface of both wire and metal
- We can vectorize multiple "for" loops to a single line in python
- Also we observed that the decrease in error is very slow after 500 iterations which makes this method inefficient